National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-984

Determining Grasp Positions for a Parallel Type Robot Gripper

G. Roth

January 1986

Canada

NRCC No. 26659

86-06461/

Copyright 1986 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1986 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

DETERMINING GRASP POSITIONS FOR A PARALLEL TYPE ROBOT GRIPPER

G. ROTH

ERB-984

JANUARY 1986

Contents

Abstract/Résumé v

1 Introduction 1

2 Computational Geometry 1
- 2.1 Preliminaries . . . 1
- 2.2 Growing Convex Polygons . . . 2
- 2.3 Growing Monotone Polygons . . . 3
- 2.4 Intersecting Monotone Polygons . . . 7
- 2.5 Sets of Polygons . . . 9
- 2.6 Parallel Visiblitiy . . . 10

3 Grasping Applications 10
- 3.1 Using a Geometric Model . . . 10
- 3.2 Using Sensor Data . . . 13
- 3.3 Handling Uncertainty in Location . . . 15

4 Conclusion 16

List of Figures

1 Growing a convex polygon by a convex polygon . . . 3
2 A polygon monotone in the x direction . . . 4
3 A polygon not monotone in any direction . . . 4
4 A grown top chain . . . 5
5 A grown bottom chain . . . 5
6 Merging two grown chains . . . 6
7 A monotone polygon grown by a convex polygon . . . 7
8 A parallel gripper on an industrial robot . . . 11
9 A part to be grasped . . . 12
10 Acceptable gripping polygons from a geometric model . . . 14
11 Gripper cross section . . . 15
12 Acceptable gripping polygons from sensor data . . . 16

ABSTRACT

We describe the application of techniques from computational geometry to produce algorithms that determine where a part should be grasped by a robot gripper. These algorithms work from a geometric model of the part or, if this is not available, from the part geometry as supplied by a three-dimensional sensor. They transform the three-dimensional grasping problem to two dimensions using heuristics and then solve the problem exactly in two dimensions.

The advantages of these algorithms are two-fold. First, they run in low order polynomial time complexity and, second, they produce all the acceptable grasp points. This means we can compare the grasp points to find the one that is the most stable and the most tolerant to uncertainty in the part location. The algorithms are described in pseudo-code and are illustrated by a number of examples.

RÉSUMÉ

On décrit dans le présent document l'application du calcul géométric à la production d'algorithmes qui déterminent comment une pièce est saisie par une pince de report. Ces algorithmes travaillent à partir d'un modèle géométrique de la pièce ou, s'il n'existe pas de tel modèle, à partir de données fournies par un capteur tridimensional. Ils ramènent au moyen de méthodes heuristiques le problème tridimensional de la préhension à un problème bidimensionnel dont ils déterminent ensuite la solution exacte.

Ces algorithms présentent deux avantages. Tout d'abord, ils font appel à des fonctions polynominales temporelles d'ordre peu élevé. Ensuite, ils déterminent tous les points acceptables de préhension. Il est donc possible de comparer tous ces points et de choisir celui qui donne la meilleure stabilité et qui est le moins sensible à l'incertitude de la détermination. Les algorithmes sont décrits en psuedo-code et illustrés au moyen d'un certain nombre d'examples.

DETERMINING GRASP POSITIONS FOR A PARALLEL TYPE ROBOT GRIPPER

G. ROTH

1 Introduction

Currently the programming of robots is a time consuming and tedious process even with the use of high level languages like VAL. The reason is that small deviations in the geometry of parts and their location make drastic differences in the correctness of a program. Ideally, the robot program should be generated automatically from a description of the task and modified appropriately by sensors to compensate for uncertainty in the environment. In this way robot programming would be done at a "task" level, which is the level at which people work, as opposed to the "manipulator" level, which is the level at which robots function.

There are a number of problems that must be solved before task level programming becomes a reality. The one we concentrate on is the grasping problem, that is, determining how a parallel type robot gripper should grasp an object given one of the following scenarios.

1. A geometric description of the part is available and from this description a number of grasp points must be generated along with their approach vectors.

2. No geometric description of the part exists but a three-dimensional sensor is available and from the information provided by this sensor a number of grasp points must be generated along with their approach vectors.

The restrictions we place on the robot motion is that the gripper must move to the grasp point in a single motion along the approach vector and that this approach vector must be parallel to the gripper orientation. In terms of the robot language VAL the robot must move to the grasp point in one straight line motion in the direction of the tool orientation.

2 Computational Geometry

2.1 Preliminaries

This is a new discipline which combines classical geometry with an analysis of computational complexity (1). The objectives of this discipline are to create simple, efficient, and provably correct algorithms for certain basic operations. An example of such an operation is to determine whether any two of a set of line segments intersect. Once these basic algorithms exist more complex ones can be created by using them as subroutines.

We will show how this technique is applied in our situation, but first we need a number of definitions. The methods that we describe are two-dimensional in nature and we will show in Section 3 how they are applied to three-dimensional robot problems.

Definition 2.1.1 *An n vertex polygon is defined by its vertices* $(p_1, \ldots, p_n)$ *listed in clockwise order.*

Definition 2.1.2 *A polygon is simple if no two non-consecutive sides intersect and connecting sides intersect only at a single point.*

Definition 2.1.3 *A polygon is convex if the angle between consecutive sides is less than π radians.*

Definition 2.1.4 *A polygon is in standard form if all its vertices are listed in clockwise order, no three vertices are collinear, and the vertex labelled p_1 has maximum y co-ordinate (in the case of a tie choose the vertex with the minimum x co-ordinate).*

2.2 Growing Convex Polygons

Spatial planning is an important and growing area of computational geometry with applications in robotics and VLSI layout. In general, spatial planning involves reasoning about the motion or placement of an object given some geometric constraints. Determining grasp positions is a spatial planning problem and the techniques that have been developed in this field are applicable. The basic operation that we use to solve the grasping problem is a "grow" operation which we will now define more formally. Assume we are given two simple polygons, P and Q. Polygon P has m vertices and is fixed, polygon Q has n vertices and can be translated, but not rotated.

Definition 2.2.1 *Vertex one of polygon Q is defined as the reference vertex of Q since its position completely defines the position of all the other vertices of Q.*

Definition 2.2.2 *A "grow" operation takes the polygon P and grows it by polygon Q to create a polygon $P_{Grown-Q}$ containing the fixed polygon P. It has the characteristic that if the reference vertex of Q is outside the polygon $P_{Grown-Q}$ then Q does not intersect P.*

If both P and Q are convex with P having m vertices and Q having n vertices then there exists (2) an $O(m+n)$ algorithm to compute $P_{Grown-Q}$. We now summarize the algorithm in pseudo-code whose format should be self explanatory. The only exception is that comments have the form {...*Comments*...}.

```
Name: Grow
Input: A convex polygon Q that can be translated and a convex polygon P
       which is fixed.
Output: A convex polygon PGrown-Q which contains polygon P.

Reflect polygon Q about its  reference vertex and translate both polygons
  so that this reference vertex is the new origin.  { see (2) }
Grown-Polygon:= Sum (Modified Polygon P, Modified Polygon Q)
Return the Grown-Polygon after it has been translated so that
  the old origin is restored.

Name: Sum (A subroutine)
Input: Two convex polygons, one called self and the other
       called addend.
Output: A polygon which is the "sum" of the two.
```

```
Current-Vertex of self := 0.
Current-Vertex of addend := 0.
DO
  COLLECT the vector sum of both vertices as a new vertex of the sum polygon.
  Compute the angular difference between the edges at the current vertices.
  CASE Angular Difference
    0  { self = addend}
      Advance current vertex of self and addend.
    Negative { self < addend}
      Advance current vertex of self.
    Positive { self > addend}
      Advance current vertex of addend.
  UNTIL No more vertices in both polygons.
RETURN the sum polygon vertices.
```

An example of this operation is shown in Fig. 1 where the large polygon is P, the small one is Q and the dashed polygon is $P_{Grown-Q}$.

Figure 1: Growing a convex polygon by a convex polygon

2.3 Growing Monotone Polygons

We will expand this growing operation to include another class of polygons, those which are called monotone polygons. This class of polygons is a subset of simple polygons and a superset of convex polygons. Fast algorithms exist for detecting and manipulating such polygons and, as we will show, they occur naturally in our grasping problem.

Definition 2.3.1 *A chain Ci,j is a sequence of vertices $p_i,\ldots,p_j$ of polygon P.*

Definition 2.3.2 *A chain is monotone with respect to a line l if the projection of p_k, $k = i,\ldots,j$ on l are ordered in exactly the same way as the vertices of Ci,j.*

Definition 2.3.3 *A polygon is monotone if there exists a line l such that the boundary of P can be partitioned into two chains, Ci,j and Cj,i, each of which is monotone with respect to l.*

From now on we will assume that all monotone polygons and chains are monotone in the x direction. If this is not true they can in linear time (this means time proportional to the number of vertices) be rotated until they are monotone in the x direction. Then they are operated on by our algorithms and can in linear time be rotated back to their original direction of monotonicity. Since none of these algorithms have a time complexity better than linear the time complexity is not made worse by any of these rotations. An example of a polygon monotone in the x direction is shown in Fig. 2 while one that is not monotone in the x direction or in any other direction is shown in Fig. 3.

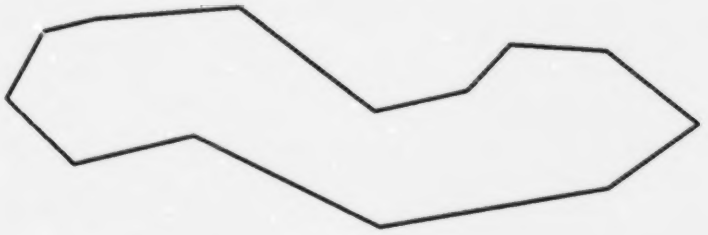

Figure 2: A polygon monotone in the x direction

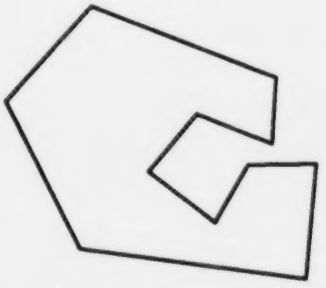

Figure 3: A polygon not monotone in any direction

We now describe an algorithm for computing $P_{Grown-Q}$ if P is monotone and Q is convex. However, first we require some preliminary definitions and lemmas.

Definition 2.3.4 *We are given a polygon P monotone in the x direction with extreme vertices i and j, where i is the vertex with the minimum x coordinate and j is the vertex with the maximum x coordinate. Then the polygon can be divided into two chains, Ci,j and Cj,i. Chain Ci,j is called the top chain and chain Cj,i is called the bottom chain since this is the relationship the chains have by their y coordinate (remember that the polygon is in standard form and vertices are listed in clockwise order).*

Definition 2.3.5 *For the top chain Ci,j the extended polygon P_Extended(Ci,j) is defined by the vertices $Ci,j,(Pj_{xcoord},-\infty),(Pi_{xcoord},-\infty)$. Similarily, for the bottom chain Cj,i the extended polygon P_Extended(Cj,i) is defined by the vertices $Cj,i,(Pi_{xcoord},+\infty),(Pj_{xcoord},+\infty)$.*

Definition 2.3.6 *We define $Ci,j_{Grown-Q}$ and $Cj,i_{Grown-Q}$ as follows. Take Ci,j and make a polygon P_Extended(Ci,j) from it and compute P_Extended$(Ci,j_{Grown-Q})$. The top chain of this*

polygon is defined as $Ci, j_{Grown-Q}$. *Similarily take* Cj,i *make a polygon P_Extended*(Ci,j) *from it and compute P_Extended*$(Cj, i_{Grown-Q})$. *The bottom chain of this polygon is defined as* $Cj, i_{Grown-Q}$.

An example of a grow operation of a top chain by a convex polygon is shown in Fig. 4, and of a bottom chain by a convex polygon is shown in Fig. 5 with the grown chains in dashed lines.

Figure 4: A grown top chain

Figure 5: A grown bottom chain

Lemma 2.3.1 *Given a polygon P monotone in the x direction and a convex polygon Q then* $P_{Grown-Q}$ *consists of two chains. The top chain is* $Ci, j_{Grown-Q}$ *and the bottom chain is* $Cj, i_{Grown-Q}$ *where i and j are the extreme vertices of the polygon as previously defined.*

Proof: The polygon can be divided into two chains Ci, j and Cj, i since it is monotone. Therefore $P_{Grown-Q}$ is created by growing both these chains by Q and then connecting them at their extreme vertices.

Definition 2.3.7 *A convex monotone chain* Ci, j *is a monotone chain where the angles the edges make at vertices* $P_{i+1}, \ldots, P_{j-1}$ *are all less than* π *radians.*

We describe the algorithm used to grow the top and bottom chains. In particular, we describe only the growing operation on the top chain since the bottom chain is similar. The method used is the one suggested (2) for growing polygons that are not convex and it is to divide the polygon into convex chains, grow each chain and combine the results. It is not clear how to combine the grown chains for the general case of a simple polygon but there is a way to accomplish this task for a monotone polygon.

```
Name: Grow Chain
Input: A top chain Ci,j of a monotone polygon Q that can be translated
       and a convex polygon P which is fixed.
Output: A grown top chain, Ci,j-Grown-Q which contains polygon P.

Current grown chain stack := NIL.
Current chain vertex := i.
WHILE Current chain vertex \= j
DO
  Get the next convex monotone chain starting from the current chain vertex.
  Grow the convex monotone chain by the convex polygon Q
    using the standard convex grow routine.
  From right to left find the intersection point of the grown convex
    chain with the line segments on the current grown chain stack.
  Pop all the vertices of the current grown chain to the right of
    this intersection point.
  Push the intersection point and all the vertices of the grown
    convex chain to the right of the intersection point on the
    current grown chain stack.
  Advance the current grown chain vertex to the end of the current
    grown chain.
FINALLY Return the reverse of the vertices on the current grown chain
        stack as the grown chain.
```

Lemma 2.3.2 *Consider two monotone chains Ci,j and Cj,k which connect at vertex j. Then $Ci,j_{Grown-Q}$ and $Cj,k_{Grown-Q}$ intersect at exactly one point and $Ci,k_{Grown-Q}$ consists of the portion of $Ci,j_{Grown-Q}$ to the left of this intersection point, the intersection point, and the portion of $Cj,k_{Grown-Q}$ to the right of the intersection point.*

Proof: Consider Fig. 6 where polygon Q is positioned to contact both Ci,j and Cj,k. The small circles are the contact points, the large circle is vertex j and the left and rightmost vertices are labelled i and k, respectively. Because of the convexivity of Q no part of Ci,j to the right of the leftmost contact point can contact Q without Q intersecting some part of Cj,k. The same argument holds for any part of Cj,k to the left of the rightmost contact point. Therefore the position of the reference vertex of Q in the diagram is the intersection point of $Ci,j_{Grown-Q}$ and $Cj,k_{Grown-Q}$.

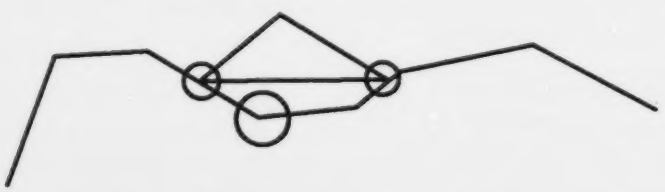

Figure 6: Merging two grown chains

Lemma 2.3.3 *If chain $Ci, j_{Grown-Q}$ has n1 vertices and chain $Cj, k_{Grown-Q}$ has n2 vertices this intersection point can be found in $O(n1+n2)$ time. Therefore $Ci, k_{Grown-Q}$ can also be found in the same order time given that $Ci, j_{Grown-Q}$ and $Cj, k_{Grown-Q}$ already exist.*

Proof: Simply merge $Ci, j_{Grown-Q}$ and $Cj, k_{Grown-Q}$ from right to left till the first intersection point is found since by Lemma 2.3.2 we need not search any farther. The part of $Ci, j_{Grown-Q}$ to the right of this intersection point and the part of $Cj, k_{Grown-Q}$ to the left of this intersection point can therefore be discarded. The merge operation can be done in linear time and the discarded parts of the chains need never be considered again.

Lemma 2.3.4 *If an upper (or lower chain) contains n vertices with k convex chains and polygon Q contains m vertices then chain can be grown by Q in $O(km+n)$ time.*

Proof: Say convex chain number i has r_i vertices. Then the growing operation takes $O(m+r_1+ ... + m+r_k)$ time. But $\sum_{i=1}^{k} r_i = n$ so this expression can be simplified to $O(km+n)$ which is the maximum number of vertices possible in the grown chain. Since the grown convex chains can be merged in time proportional to the total number of vertices in these chains the time complexity of the merge operation is also $O(km+n)$.

Theorem 2.3.1 *A monotone polygon P with n vertices can be grown by a convex polygon Q with m vertices in $O(km+n)$ time where k is the number of convex chains in the monotone polygon. The resulting polygon is monotone in the same direction as P.*

Proof: By Lemma 2.3.4 the top and bottom chains can be grown in $O(k1m+n)$ and $O(k2m+n)$ time where $k1$ and $k2$ are the number of convex chains in the top and bottom chains. Since $k = k1 + k2$ and by Lemma 2.3.1 the grown monotone polygon consists of the grown top and bottom chains the conclusion is $P_{Grown-Q}$ can be computed in $O(km+n)$ time.

An example of this growing operation is shown in Fig. 7 where the grown polygon is drawn in dashed lines. The time complexity of the grow operation can be reduced somewhat (3) but for small k and m the difference is not significant.

Figure 7: A monotone polygon grown by a convex polygon

2.4 Intersecting Monotone Polygons

The operation of intersecting two polygons will be important for our application, therefore we want to find efficient and conceptually clear algorithms to accomplish this task. Assume we are given two simple polygons P and Q where P has n vertices and Q has m vertices.

Lemma 2.4.1 *If P and Q have no other restrictions on them other than being simple polygons their intersection may take $O(mn)$ time to compute.*

Proof: Simply draw P and Q as "star"-shaped polygons that intersect maximally. In this case there may be up to mn polygons created by the intersection process so the order must be $O(mn)$.

Lemma 2.4.2 *If P and Q are both convex they can be intersected in $O(m+n)$ time using the slab method (1) and the result is a convex polygon.*

Proof: The basic idea of the slab method is to intersect vertical slabs of each polygon from left to right and compute the intersection polygon by taking the union of these slabs. At each step the two vertical slabs consist of the quadrilateral subparts of each polygon which are between two vertical lines defined by consecutive vertices of both polygons in the x direction. Since there are at most $m+n$ such slabs and the slabs are quadrilaterals that can be intersected in fixed time the intersection process takes $O(m+n)$ time. Each quadrilateral is convex so the intersection of these polygons is also a convex polygon.

Theorem 2.4.1 *Two polygons monotone in the same direction can be intersected in $O(m+n)$ time and the results are $O(m+n)$ polygons that are also monotone in this direction.*

Proof: Rotate both polygons till they are monotone in the x direction. Now use the slab method for intersecting these polygons. Then rotate the intersection polygons so that they are monotone in the original direction to get the final result. The only difference between this and the convex case is that the slabs may be concave quadrilaterals and the result is O(m+n) polygons instead of just one polygon.

The pseudo-code for the algorithm is as follows.

```
Name: Intersect two polygons monotone in the x direction.
Input: Two polygons
Output: A set of polygons monotone in the x direction which are the
        intersection of these two polygons.

Get top and bottom chains of both polygons as a collection of line segments
  ordered from left to right.
Start of slab is maximum of the X start of first segment of all the chains.
WHILE All chains have segments still to be processed.
DO
  End of slab is the x minimum of current segment endpoints.
  Compute vertices of both quadrilaterals in the slab.
  Compute the intersection of both quadrilaterals.
  Add the intersection points to the current intersection polygon.
  IF the intersection polygon is terminated save it.
  Advance a polygon chain if its' current segment ends the slab.
  Set the start of the slab to the current end of the slab.
RETURN The Intersection Polygons.
```

The method of intersecting quadrilaterals is a simple one. The two vertical segments (labelled first and second) at the left and right ends of each quadrilateral can be related horizontally in six ways which are as follows:

First segment is above and disjoint from the second
First segment contains the second
First segment is above and intersects the second
First segment is below and is disjoint from the second
First segment is contained in the second
First segment is below and intersects the second

This produces thirty-six possible cases for the intersections of the quadrilaterals and they can easily be enumerated. This means the quadrilaterals can be intersected in fixed time.

2.5 Sets of Polygons

A set S is a set of n convex polygons $Q_1, \ldots, Q_n$ where the position of each polygon relative to every other polygon is fixed. This means that the entire set moves together as one and the location of each polygon in the set is known given the reference point of the set which we call S_r. The reference point can be chosen arbitrarily to be any point contained in any of the polygons in the set. Assume P is a fixed polygon and that the set can be translated, but not rotated when it is moved.

Definition 2.5.1 *$P_{Grown-Any-S}$ is a polygon (or polygons) which have the characteristic that if S_r is contained in any of these polygons then some polygon in S intersects P.*

Definition 2.5.2 *$P_{Grown-All-S}$ is a polygon which has the characteristic that if S_r is contained in this polygon then every polygon in the set S intersects P.*

The pseudo-code for the algorithm to compute the translated grown set is as follows.

```
Name: Compute translated grown set
Input: Polygon P and the set S
Output: The translated grown set S'

FOR Each Q in S
  Compute P-Grown-Q.
  Translate it by the vector difference between Sr and
    the reference vertex of Q.
  COLLECT the translated P-Grown-Q.

RETURN S' which is {Translated P-Grown-Q1,...,Translated P-Grown-Qn}.
```

Lemma 2.5.1 *$P_{Grown-Any-S}$ consists of Translated $P_{Grown-Q_1} \cup \ldots \cup$ Translated $P_{Grown-Q_l}$.*

Proof: If S_r is in any of the translated grown polygons then it meets the conditions that define $P_{Grown-Any-S}$ so we need simply take the union of all these polygons.

Lemma 2.5.2 *$P_{Grown-All-S}$ consists of Translated $P_{Grown-Q_1} \cap \ldots \cap$ Translated $P_{Grown-Q_l}$.*

Proof: If S_r is in all of the translated grown polygons then it meets the conditions that define $P_{Grown-All-S}$ so we need simply take the intersection of all these polygons.

2.6 Parallel Visiblitiy

Another problem that must be solved for our application is that of parallel visibility. In this model of visibility the viewpoint is located at infinity and oriented in a given direction.

Definition 2.6.1 *Imagine a light source at the viewpoint emitting parallel light rays in a given direction. The parallel visibility chain of a set of objects is the part of the objects that is illuminated by this light source. It is a chain C which is monotone in a direction perpendicular to the direction of the rays from this light source.*

Lemma 2.6.1 *For a polygon with n vertices the parallel visibility chain can be computed in $O(n)$ time.*

Proof: Transform the parallel visibility problem to a perspective visibility problem, use Lee's algorithm (4) to compute the visibility chain, and reverse the transformation to produce the parallel visibility chain.

Lemma 2.6.2 *Given n line segments in the plane which intersect at s points the parallel visibility chain can be found in $O((n+s)\log n)$ time.*

Proof: The intersection points can be found in this order time using a plane sweep algorithm (5). Once these points are found another plane sweep using a balanced binary tree can find the visibility chain in the same order time.

3 Grasping Applications

3.1 Using a Geometric Model

Assume we are given a geometric model of an object as a boundary representation, that is, a description of the surface that defines the boundary of a three-dimensional object. The surface can be described by a union of planar patches which are called faces. This type of representation can be displayed by such packages as MOVIE and this package was used to produce the three-dimensional diagrams in this report.

Given the geometric model of the part we want to find the grasp points and approach vector of parallel robot gripper. The assumptions we make were described in Section 1 and we repeat them here.

1. The gripper must approach the grasp point in one straight line motion along the approach vector and must be oriented in the same direction as this vector.

2. The gripper is a standard parallel gripper which has a top part joining two jaws that close together to grasp a part.

A diagram of a parallel gripper connected to a typical industrial robot is shown in Fig. 8. Its orientation is defined to be a vector parallel to the largest side of the gripper jaws.

The method used is to transform the three-dimensional problem to a two-dimensional one and then to solve the two-dimensional problem using the techniques of computational geometry as described in Section 2. The algorithm to transform from a three-dimensional to two-dimensional problem is the one described by Wolter (6).

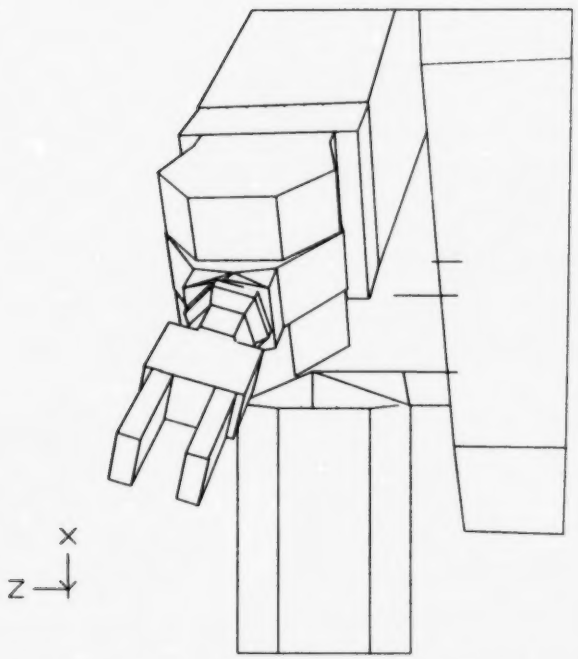

Figure 8: A parallel gripper on an industrial robot

1. Each face of the object consists of the vertices of a polygon in a plane which has an equation of the form $Ax + By + Cz + D = 0$. The vector (A, B, C) is the normal to the plane and D is the distance from the origin to the plane. Sort all the planes by their distance from the origin and find all pairs of planes $(P1, P2)$ such that $\|D1 - D2\| <= W$ where W is the maximum gripper width and the normals $(A1, B1, C1)$ and $(A2, B2, C2)$ are within some δ of being within the opposite direction from each other. Such pairs of planes are called potential grasp planes.

2. For each pair of potential grasp planes determine the intersection of the polygons in these planes and call the polygon which is their intersection the gripping area. If this gripping area is large enough proceed to Step 3, otherwise try the next pair of potential grasp planes.

3. Compute the swept volumes of the jaws of the gripper as the gripper moves in a path parallel to the gripping area. Project each line segment of a face that crosses or is contained in this swept volume onto the plane parallel to the gripping area.

The result of these operations is a number of line segments which are the obstacles that the gripper must not contact and the gripping area which defines the target area of the gripper. This algorithm can be repeated with a face and vertex pair (7) if no good face pairs are found. An example of a part is shown in Fig. 9 and in our example the gripper will grasp the upright back of this part in a vertical direction.

Now we have a two-dimensional problem and the remaining task is to identify grasp points and approach vectors. The algorithm we use is as follows:

1. Arbitrarily select a number of approach directions by choosing all the directions perpendicular to the edges of the gripping area and all the directions which bisect the angles between the edges of the gripping area polygon (as in ref. 6).

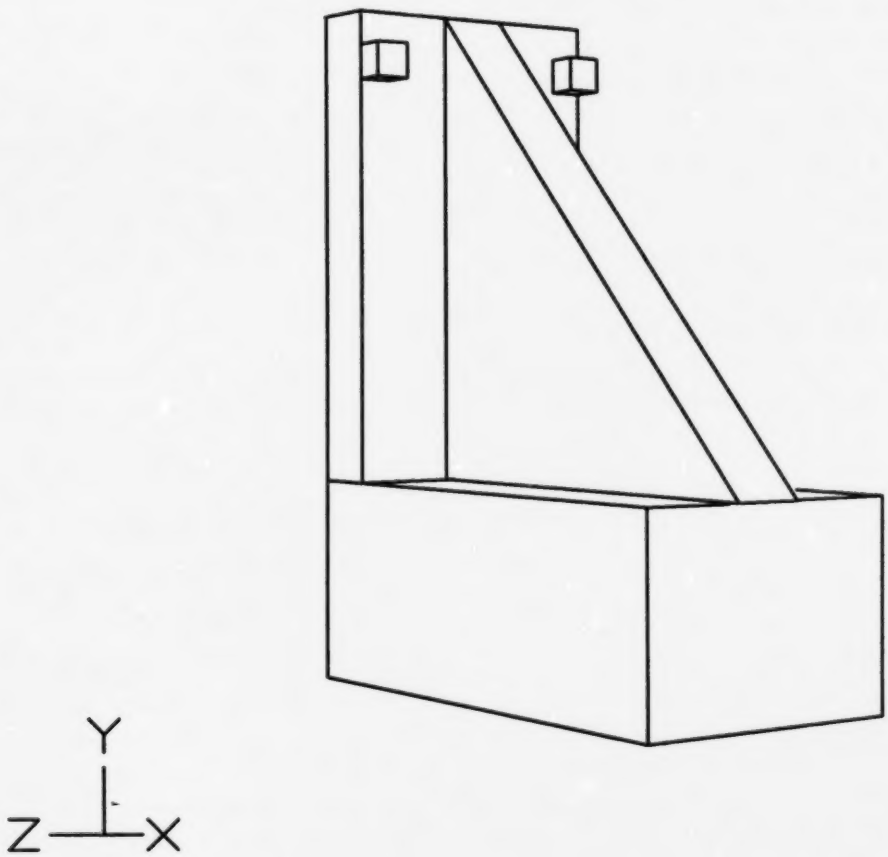

Figure 9: A part to be grasped

2. Compute the acceptable gripping polygons and find the best grasp points in each such polygon.

The concept of an acceptable gripping polygon is now defined in more detail. Assume the plane has been rotated so that the motion of the gripper is in the negative y direction. The gripper consists of two parts, a horizontal part and a vertical part. The horizontal part is the part of the gripper that connects the jaws and it is perpendicular to the plane containing the gripping area. The vertical parts represent the jaws and they are parallel to the plane containing the gripping area. When these parts are projected into the gripping plane the result is a rectangle which is the projection of the horizontal part (hereafter known as the horizontal projection) on top of the rectangle which is the projection of the vertical part (hereafter known as the vertical projection). These facts lead to the following simple definition for an acceptable grasp.

Definition 3.1.1 ***A grasp is acceptable if the gripper is in a position where neither the horizontal projection nor the vertical projection intersects the obstacles, the vertical projection intersects the gripping area, and the horizontal projection does not intersect the gripping area.***

Consider the vertical projection and horizontal projection as defining a set S of polygons called

the gripper set with reference point S_r being shown with a circle.

Definition 3.1.2 *A gripping polygon is a polygon that has the characteristic that if the reference point of the gripper set is contained in this polygon the grasp defined by the location of this point is an acceptable grasp.*

To calculate all the acceptable gripping polygons we use the following steps.

1. Perform parallel visibility in the approach direction (which is the negative y direction in our example) on the gripping area and the obstacles to produce a set of chains monotone in the x direction. Since the gripping area is a simple polygon with say n vertices the parallel visibility can be done in $O(n)$ time. For the obstacles which consist of a set of n line segments which intersect in s points the parallel visiblity can be done in $O((n+s)\log n)$ time. Both methods are described in Subsection 2.6. The result is a top chain of the gripping area called *C_Gripping_Area* and a top chain of the obstacles called *C_Obstacles*.

2. Compute $C_Grown_Gripping_Area_{Horisontal_Projection}$ and $C_Grown_Obstacles_{Horisontal_Projection}$. Then reverse $C_Grown_Gripping_Area_{Horisontal_Projection}$ and take the extended polygon of the bottom chain and call it $P_Extended(C_Reversed_Grown_Gripping_Area_{Horisontal_Projection})$. In the same fashion compute $P_Extended(C_Reversed_Grown_Obstacles_{Horisontal_Projection})$ and repeat this process on both chains with the vertical gripper projection in place of the horizontal gripper projection.

3. Each of the four extended polygons is now translated by the vector difference between their reference point and the reference point Sr. Then the acceptable gripping polygon can be computed by

$$\begin{aligned}
&Translated\ P_Extended(C_Reversed_Grown_Gripping_Area_{Horisontal_Projection})\cap\\
&Translated\ P_Extended(C_Reversed_Grown_Obstacles_{Vertical_Projection})\cap\\
&Translated\ P_Extended(C_Reversed_Grown_Obstacles_{Horisontal_Projection})\cap\\
&Translated\ P_Extended(C_Grown_Gripping_Area_{Vertical_Projection})
\end{aligned}$$

   Each of the growing operations is done by the convex grow of a monotone polygon and the intersections are done by the slab method.

4. In each of the acceptable gripping polygons find the best final gripper location by a heuristic combination of closeness to center and depth.

These ideas are made clearer by Fig. 10 where the gripping area is the large rectangle, the obstacles are the small rectangles, the acceptable gripping polygons are in dashed lines and the best gripping locations are drawn in circles. This completes the process since the best gripping locations defines the grasp positions and we have already computed the approach vector.

3.2 Using Sensor Data

Assume there exists a three-dimensional rangefinder as described in ref. 8 that returns a standard pixel image but instead of the pixel value being proportional to light intensity it is proportional to the distance from the viewpoint of the rangefinder. Now consider a number of randomly oriented

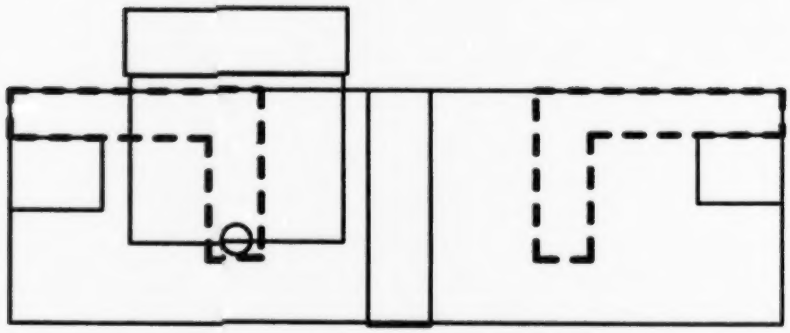

Figure 10: Acceptable gripping polygons from a geometric model

parts which may be overlapped and assume we want to grasp one part at a time from this jumble of parts. Given an image provided by a rangefinder and a robot with a parallel gripper which acts under the same restrictions as the previous section we want to find the grasp points and the approach vectors to pick out these parts. This problem is called the bin picking problem and a number of techniques have been described to solve this problem (9). However, these techniques are more suited to intensity images and do not make use of all the information provided by range data.

In order to solve the problem we use the same method as before, that is, to transform the problem from a three-dimensional to a two-dimensional problem, then solve the two-dimensional problem using our computational geometry techniques. To transform the problem from three dimensions to two dimensions simply consider a vertical slice of the parts and a vertical slice of the gripper. The gripper is a standard parallel gripper and a single vertical slice is shown in Fig. 11. The objective is to find to what extent the gripper fits the object to be grasped in the vertical slice and then to find the best of the grasps from this information. This process is called structure mating and our algorithm produces the same results as in the literature (10) but for two dimensions instead of three. The structure mating process requires morphological operations whose running time is a function of the size (number of pixels) in the structuring elements (equivalent to our moving polygons) which makes these algorithms very slow unless high speed hardware is used. However, our algorithm has a running time which is a function of the number of segments in the moving polygons, not the size, and are thus efficient enough to run in a relatively short time on standard hardware. We assume as before that the gripper moves in a single translation in the vertical direction to the grasp point. In order to formalize these ideas we make the following definitions.

Definition 3.2.1 *The gripper consists of a set of three rectangles which move together and are called the non-contact members. The parallel jaws of the gripper close during a grasp attempt and we define the rectangle that is swept by the jaws as the contact member.*

Definition 3.2.2 *A grip is acceptable if no part of any non contact member intersects any part of the object and some part of the contact member intersects some part of the object.*

Definition 3.2.3 *Given a reference point of the gripper we define a gripping polygon as one having the characteristic that if the reference point is contained in this polygon then the grip defined by the location of this reference point is acceptable.*

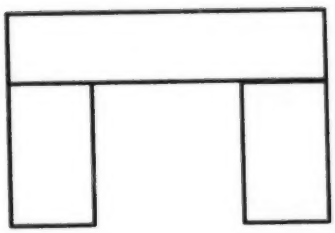

Figure 11: Gripper cross section

Of course we may inadvertantly pick up two objects instead of one but this is unlikely. We want to find all the acceptable gripping polygons for a vertical slice, which we do as follows:

1. Compute the vertices of a polygonal chain C which is monotone in the x direction from the sensor pixels using a divide and conquer method (11).

2. Compute the extended polygon of the reverse of this chain and grow it by each of the non-contact members to produce $P_Extended(C_Reverse_{Non_Contact_Members})$. Then compute the extended polygon of this chain and grown it by the contact member to produce $P_Extended(C_Contact_Member)$.

3. Each of the two extended polygons is now translated by the vector difference between their reference point and the reference point of the gripper. The acceptable gripping polygon consists of

   $$Translated\ P_Extended(C_Reverse_{Non_Contact_Members}) \cap$$
   $$Translated\ P_Extended(C_Contact_Members)$$

4. In each of the acceptable gripping polygons find the best location by a heuristic combination of depth and closeness to vertical of a pair of opposing lines of the polygon (for stability).

An example is shown in Fig. 12 where the acceptable gripping polygons are shown in dashed lines and the gripper reference point is drawn in a circle.

This completes the process for one slice which is the two-dimensional case. For the three-dimensional case divide the gripper into r vertical slices and find the acceptable gripping polygons by the methods described. Intersect these polygons for each slice to find the final acceptable gripping polygons for that location of the three-dimensional gripper (there may not be any). Intersecting all the acceptable gripping polygons in each of the r slices can be done using a divide and conquer technique in $O(r \log r)$ time where at each step we intersect the acceptable gripping polygons for that set of slices. Note that if we are scanning an image till we find the first r acceptable grips then we can in $O(\log r)$ time add a new slice and delete an old one by using the divide and conquer tree. Thus, if we have d total slices and we search for the first r acceptable grips then we need $O(d \log r)$ time to accomplish the intersections.

3.3 Handling Uncertainty in Location

The main advantage of these methods over other algorithms to find grasping points is that all the acceptable gripping positions are found since these are defined as the acceptable gripping polygons.

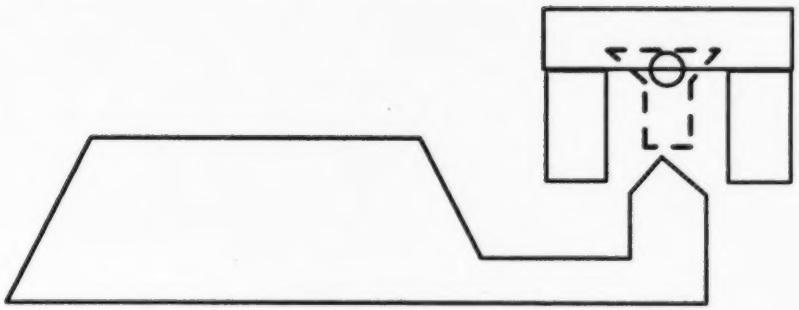

Figure 12: Acceptable gripping polygons from sensor data

This allows us to use these polygons to estimate the translational uncertainty that can be tolerated by the chosen grasp.

Definition 3.3.1 *Assume we are given a grasp location which is defined by a vector $\overline{P}$ with its' endpoint in an acceptable gripping polygon. Also assume we are given a direction α in which the part can be translated. Then the grasp is acceptable as long as the part is translated by a distance l or less, where this is the distance from the endpoint of $\overline{P}$ to the closest edge of the gripping polygon. The grasp is said to have a tolerance to translational uncertainty of l in direction α.*

Definition 3.3.2 *Assume we are given a grasp location which is defined by a vector $\overline{P}$ with its' endpoint in an acceptable gripping polygon. Find the largest circled centered at the endpoint of $\overline{P}$ which is entirely contained in this acceptable gripping polygon. The radius of this circle d is the distance that the part can be translated in any direction and continue to meet the constraint that the grasp is acceptable. The grasp is said to have a tolerance to translational uncertainty of d in any direction.*

The calculation of these parameters that describe that acceptability of the grasp is simple and they are used to decide on the final grasp position by favoring grasp points that have a high tolerance to translational uncertainty of the part. Unfortunately, rotational uncertainty is more difficult to handle because a rotation affects the visibility chain of the part and therefore creates new acceptable gripping polygons.

4 Conclusion

We have presented a new algorithm that performs growing operations on monotone polygons and shown how it can be applied to some problems in the generation of grasp points. This method has the advantage over the techniques of ref. 7 in that it finds all the acceptable gripping polygons and uses the best gripping points in them and does this in bounded polynomial time instead of using an unbounded number of comparisons as in ref. 6. It works both when a geometric model exists of the part and when the part geometry is obtained from a rangefinder. Therefore the algorithm is useful in the area of task planning and in the processing of range data for object manipulation.

REFERENCES

1. Shamos, M.I. ***Computational Geometry.*** Ph.D. dissertation, Yale Univ., New-Haven, CT. 1978.

2. Lozano-Perez, T. Spatial planning; configuration space approach. *IEEE Trans. Comput.* 32(2): 108-120; 1983.

3. Roth, G. Growing and shrinking operations on monotone polygons. *Proc. 23 Allerton Conference on Communications and Control*, Urbana, Illinois. October 1985. pp. 508-510.

4. Lee, D.T. Visibility of a simple polygon. *Computer Vision, Graphics and Image Processing* 22(4): 207-221; 1983.

5. Melhorn, K. ***Multi-dimensional Searching and Computational Geometry.*** Springer-Verlag, New-York. 1984.

6. Wolter, J.D. Automatic generation of gripping positions. *IEEE Trans. Syst. Man Cybern.* 15(2): 204-213; 1985.

7. Laugier, C. A program for automatic grasping of objects with a robot arm. *Proc. 11th Symp. on Industrial Robots*, Tokyo, Japan. 1981. pp. 287-294.

8. Rioux, M. Laser range finder based on synchronized scanners. *Appl. Opt.* 23(21): 3837-3844; 1984.

9. Kelly, R. Three vision algorithms for aquiring workpieces from bins. *IEEE Proc., Special Issue on Robotics.* 71(7): 803-820; 1983.

10. Coleman, E. Aquisition of randomly oriented workpieces through structure mating. *Proc. 9th Int. Conf. Pattern Recogn.*, San-Fransisco, California. June 1985. pp. 350-357.

11. Pavlidis, T. ***Algorithms for Graphics and Image Processing.*** Computer Science Press, Rocville, MD. 1981.